Legacy systems: managing evolution through integration in a distributed and object-oriented computing environment

David Lemaitre, Dominique Sauquet, Isabelle Fofol, Loïc Tanguy,
François-Christophe Jean, and Patrice Degoulet

Medical Informatics Department, Broussais University Hospital, Paris

Legacy systems are crucial for organizations since they support key functionalities. But they become obsolete with aging and the apparition of new techniques. Managing their evolution is a key issue in software engineering. This paper presents a strategy that has been developed at Broussais University Hospital in Paris to make a legacy system devoted to the management of health care units evolve towards a new up-to-date software. A two-phase evolution pathway is described. The first phase consists in separating the interface from the data storage and application control and in using a communication channel between the individualized components. The second phase proposes to use an object-oriented DBMS in place of the homegrown system. An application example for the management of hypertensive patients is described.

INTRODUCTION

Bennett[1] defines legacy systems as "large software systems that we don't know how to cope with but that are vital to our organizations". Improving the value of these systems is of great concern. Because they have been neglected or because no one dared to modify them, they are now out-of-date programs, although they perform crucial work. Furthermore, they represent years of accumulated experience and knowledge[1].

In addition to these internal problems, the external world in which these pieces of software are running has changed considerably. This is especially true in medical care computing. Important organizational and technical changes have occurred.[2] Decision making can benefit from numerous and various information and knowledge sources, many of them accessible electronically. New software systems must take all these changes into account and enable developers to include new facilities in their applications. But new software systems will eventually become the legacy systems of tomorrow.[1] This is why they have to be designed to be as easy to maintain and to adapt as possible. Object-orientation is widely considered to be a technology that offers the mechanisms (e.g., reuse and specialization) to meet these requirements.

Legacy systems issues have been addressed only in the last few years. Many reasons have led to this fact: researchers were not inclined to study such cases, developers do not like to do the so-called "maintenance" work which is considered less gratifying than working on new software, and maintenance costs have dramatically increased during the last decade.[1] Case studies have been published[3], tools[4] are available in some areas of reengineering, and scientific papers[5] are setting the basics for future work. Reengineering legacy systems is generally admitted to be a three-step process:[6]

- describing and analyzing the current system (legacy system),
- specifying the desired system (future system),
- and designing the pathway between the current and the desired systems.

The choice of the transition path is a software engineering process as well as a business decision. Engineers have to set up different valid solutions and managers have to choose the best solution on cost/effectiveness ratio.[7] Classical approaches[5,6] for the transition path are the following:

- *big bang* approach

  In this case, the new system is built up and put into operation while the legacy system is shut down.
- *phase-out* approach (or incremental development)

  The future system is built step by step and progressively replaces the legacy system. A two-way data translation is needed.
- *phase-in* approach (or evolutionary development)

  The legacy system is rewritten to be able to include changes. New modules are then added as required.

0195-4210/95/$5.00

This article deals with a migration experience that is currently taking place in Broussais University Hospital in Paris. The current legacy system for the management of patient medical records is described as well as the desired system. The evolution and the initial results of this experience are described and discussed.

PROBLEMS AND GOALS

The current legacy system

LIED[8] is a homegrown medical application development environment. It is an all-in-one temporal database management system with a 4th generation language (4GL). The latter is used to write programs, design character-based interfaces, and generate reports. LIED has been used since 1975 to build medical applications, in particular in the hypertension and cardiovascular domains. Approximately 20 medical units in 12 different hospitals use such applications and the oldest one manages over 60000 hypertensive patient records. It addresses particular needs of medical applications such as time-related data. It has been fine-tuned for access speed and data integrity. Although the environment has been carefully maintained and documented, it falls short in some major medical care computing areas. Indeed, it was a monolithic and closed environment, the only link with the outside world being through the file system. The data model and allowed types also show the environment's age. There was no possibility to declare abstract types and easily manipulate complex objects. In any case, the character-based interface would not be able to present such data to the user.

Despite these weaknesses, the environment has been extensively used in Broussais Hospital and others to build applications for medical units. Planning the evolution of the LIED system must take into account all the data and knowledge accumulated over the years as well as the new possibilities offered by network computing.

The desired system

The goal of the target system is to improve the end-users' (i.e., physicians and nurses) applications in the following ways:

- convivial and multimedia user interface (using a standard windowing system) to improve end-user acceptance,
- abstract data types and complex object modeling (images, signals, medical concepts, delegation mechanisms, etc.) to manage more efficient and comprehensive computerized patient records,
- easy access to dedicated services dealing with decision support, image processing, natural language processing, etc.,
- local area network connectivity to facilitate access to hospital-wide databases (e.g., patient identification, laboratory results) and to avoid redundant information,
- wide area network connectivity for extended access to knowledge bases (e.g., bibliographic databases, drug and medical knowledge databases) to help the physician in his decision-making process,
- evolution capability and expendability.

The HELIOS framework

The HELIOS[9] application framework appears to be a good candidate as a development and application environment. A HELIOS application is a set of object-oriented components, connected to a communication channel, called HUB[10], and communicating through a message-passing mechanism. Two components play a particular role in the application (kernel components): the interface manager and the object-oriented information system. The interface component handles all man-machine interactions. It understands messages such as showing a form, displaying a value in a field, etc. Every user interaction is notified to the other special component: the object-oriented central repository and control component. It handles the control of the application as well as the data storage and retrieval. All other components are services components. Such components include natural language or image processing, connection services (i.e., connection to the outside world).

EVOLUTION STRATEGY

The selected evolution strategy is described in figure 1. It is a mix of the "big bang" and "phase-in" strategies. Only the part concerning the applications (i.e., how to migrate legacy applications) will be discussed.

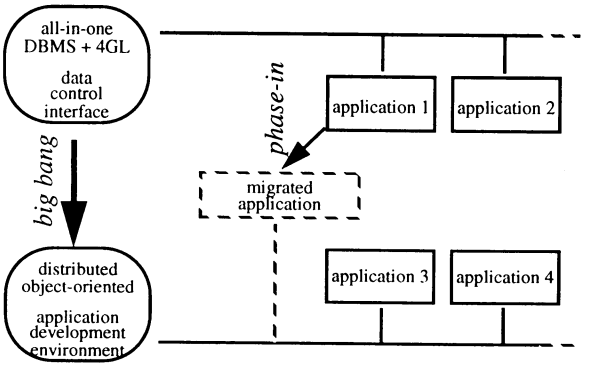

Figure 1 Evolution strategy

The migration of the legacy applications is considered as a two-phase process:

1. integrating legacy applications into the HELIOS application framework,
2. and replacing the LIED database management system by the object-oriented database management system of the new environment.

First phase

The integration phase consists in separating the interface part from the data storage and control part of the legacy system, as shown in figure 2. This creates the two base components of a standard HELIOS application that is compliant to a client/server architecture.

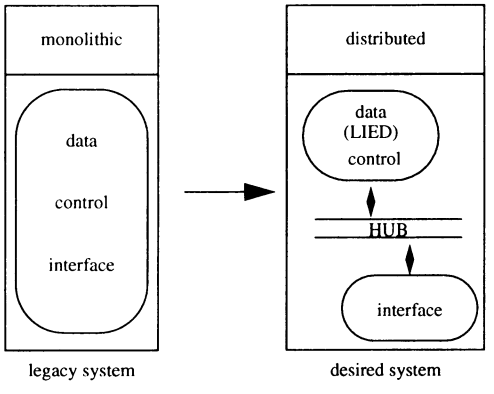

Figure 2 Migration strategy (first phase)

The interface that was included as commands in the 4th generation programming language is replaced by the HELIOS interface manager component. It consists of basic interface objects (e.g., patient selection form, alert and confirmation boxes, etc.) to which application specific objects (e.g., consultation form, biological results history, etc.) are added to create the application interface component. These objects are created with the HELIOS interface builder tool. It complies to the HELIOS style guide[12], which gives a common look and feel to every application. The data storage and control part is transformed (*phase-in* approach) into a component by extending its programming language with connection, destination, and message-passing commands. It acts as a data server and application controller.

A set of messages has been designed so that components can communicate with one another. The data storage and control component includes data storage, data retrieval, and procedure execution message classes. The interface component can understand messages such as showing or hiding a window, setting or restoring attributes (e.g., color, visibility, value, etc.) of graphical objects. All messages, coded in ASN.1, are conveyed on the HELIOS bus.[10]

This integrated application performs the same tasks as the old one. However, the ability to connect to the unification bus lets it use the service components of the environment.

Second phase

The second phase of the evolution strategy is described in figure 3. This phase mainly consists in changing the data repository. The application must be modeled in the object-oriented database. This work is facilitated by the medical concept library and data storage structures that are included as standard classes of the development environment. When the application is modeled using the environment's tools[11], the data must be exported from the legacy system to the new application.

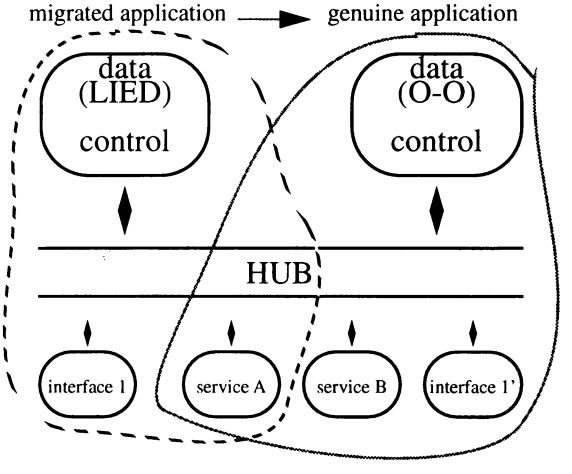

Figure 3 Migration strategy (second phase)

RESULTS

The HELIOS interface manager component and unification bus are written in C++ and run on different UNIX platforms including DEC Unix, SUN OS and Solaris, and HP-UX. The legacy system migration strategy has been tested with an application, called ARTEMIS, that deals with the follow-up of hypertensive patients, in a DEC Alpha Unix environment. The application graphical interface has been created with the HELIOS tools that are based on the X/Motif toolkit. All messages between the interface and the LIED data server component are conveyed by the unification bus. A prototype of the ARTEMIS patient repository has been developed with the GemStone object-oriented DBMS. This development has included the reengineering of the LIED data diction-

ary to a true medical concept library and the downloading of a significant set of patient records.

The ARTEMIS-2 application was installed in the outpatient clinic of the Broussais hypertensive department in January 1995. Patient data are entered directly by the physician in the presence of the patient on X/Window terminals. The traditional data entry with VT terminals was maintained in the hospital ward.

To test the migration strategy and the acceptability of the new ARTEMIS-2 application, an evaluation study was performed during a four month period from December 1994 (one month before installation) to March 1995 (three months after installation). The study concerned 9 physicians and 109 consultations at the outpatient clinic. The length of consultation, before and after the installation of the new application, were recorded. The response times were measured and the global satisfaction of the physicians was evaluated by means of visual-analog-scale based questionnaires completed by physicians at the end of the study period. Results are shown in Table 1.

Table 1. Results of the ARTEMIS-2 installation study

Evaluation criteria	Results
Before installation mean consultation duration ± SD (in minutes)	20.7 ± 7.8 (n=41)
After installation mean consultation duration ± SD (in minutes)	23.4 ± 9.4 (n=68)
Patient record retrieval time ± SD (in seconds)	3.6 ± 0.3 (n=130)
Mean degree of satisfaction ± SD (in a scale from 0 to 10)	7.9 ± 0.7 (n=9)

There was no significant increase in the consultation durations despite the minimal initial training of the physicians (less than one hour). Access to patient records was always inferior to four seconds. The mean degree of satisfaction on a scale from 0 to 10 was high (7.9). The satisfaction about the old system had not been measured in a similar way but we can notice a rise in the number of physicians willing to use the new computerized system.

DISCUSSION AND CONCLUSION

The strategy presented in this paper enables legacy systems to evolve. Integrating the legacy system, in a way very similar to the HERMES[13] project's, in a more general, open, and distributed application framework offers a much wider range of evolution possibilities. Separating the interface part from the functions part of the legacy system, as recommended by the Seeheim[14] model, enables communication (by the connection of both parts to the communication channel) and presentation (through a unified interface that could be made compliant with a medical interface style guide[12]) integration in the framework. The design of the interface, unlike Merlo[15], has been done through direct interaction with physicians to whom several prototypes have been presented. We considered that the graphical interface should not be the equivalent of the character-based one but rather should benefit from all the new possibilities. In particular, the new interface offers windows such as temperature graphs and scrollable biological ordered tables or image display.

The message-passing communication mechanism enables the legacy system to use the services offered by the application framework as any genuine application (i.e., an application developed from the ground-up with the HELIOS development environment). Thanks to this mechanism, the evolution of applications is fairly easy. It is a three step process:

- identifying the component that offers the needed service,
- getting its public interface (i.e., the messages it can answer to),
- and including these messages in the application code, where the functionality is needed.

Decision support, image and natural language processing are among the service components of the HELIOS environment that will be included in the future versions of the ARTEMIS-2 application. Components that enable applications to communicate with the outside world also exist. In particular, the Connection Services[16] component will be configured to be used as a gateway to access the central patient identity database of the hospital through EDIFACT messages. The architecture also enables a component to be replaced by a new version, provided that the set of messages (i.e., public interface) that it can understand does not change or is augmented. Applications need not be modified to take these changes into account.

ACKNOWLEDGMENTS

The development of HELIOS has benefited from the financial support of the Commission of European Union (AIM contracts A1004 and A2015) and Digital Equipment Corporation (ERP FR018).

We would particularly like to thank the physicians of the hypertension department who accepted to test the early versions of the ARTEMIS-2 application.

References

1. Bennett K. Legacy systems: coping with success. IEEE Software. 1995; 12-1: 19–23.
2. Nobel J. Changes in health care: challenges for information system design. Int. J. Biomed. Comput. 1995; 39: 35–40
3. Britcher RN. Re-engineering software: a case study. IBM Systems Journal. 1990; 29-4: 551–567.
4. Rock-Evans R., Hales K. Reverse Engineering: Markets, Methods and Tools. London, Ovum LTD. 1990.
5. Chikofski EJ., Cross JH. Reverse engineering and Design Recovery: A Taxonomy. IEEE Software. 1990; 7:1: 13–17.
6. Feiler P. Reengineering: An engineering problem. Special Report, CMU/SEI-93-SR-5. 1993. (accessible on WWW at http://www.sei.cmu.edu/SEI/pubs/abstracts/sei93sr5.html)
7. Sneed H. Planning the Reengineering of Legacy Systems. IEEE Software. 1995; 12-1: 24–34.
8. Sauquet D., Degoulet P. LIED: A Semantic and Temporal Data Management Language. Software Engineering in Medical Informatics (Ed Timmers T., Blum B.). IMIA, North-Holland, Amsterdam. 1991; pp. 267–277.
9. Jean FC., Degoulet P., Jaulent MC., et al. HELIOS: A Medical Object-Oriented Software Engineering Environment. Image Management and Communication in Patient Care (Ed Mun S., Lemke H.). IEEE Computer Society Press, Los Alamitos, California. 1993; pp. 156–163.
10. Sauquet D., Jean FC., Lemaitre D., Zapletal E., Degoulet P. The HELIOS Unification Bus: A Toolbox to develop Client/Server Applications. Comput. Methods and Programs Biomed, Suppl. 1994; 45: S13–S22.
11. Lavril M., Doré L., Zapletal E., Jean FC., Degoulet P. A Reuse Oriented Development Database: The HELIOS Object Information System. Methods and Programs Biomed, Suppl. 1994; 45: S35–S45.
12. Boralv E, Goransson B, Olsson E, Sandblad B. Usability and efficiency. The HELIOS approach to development of user interfaces. Comput. Methods and Programs Biomed, Suppl. 1994; 45: S47–S64.
13. van Mulligen E., Timmers T., Brand J., et al. HERMES: a health care workstation integration architecture. Int. J. Biomed. Comput. 1994; 34-1-4: 267–275.
14. User Interface Management and Design. Duce D., Gomes M., Hopgood F., et al. (Eds), Springer-Verlag, 1991.
15. Merlo E., Gagné PY., Girard JF., et al. Reengineering User Interfaces. IEEE Software. 1995; 12-1: 64–73.
16. Jean FC., Mascart JJ., Codaccioni A., Lavril M., Sauquet D., Degoulet P. Using a meta-model to build a Connection Service in an object-oriented medical application development environment. Proc. of the 18th SCAMC. 1994; pp. 483–487.